25 June

Branch and bound for IP [P&S §18.1]

Recall: Given an IP, the LP relaxation is the linear program obtained by removing the integer domain restrictions.

Observations

- Every feasible solution to the IP is also feasible for the LP relaxation, because the LP relaxation only removes restrictions.

- Therefore, the optimal solution to the LP relaxation cannot be worse than the optimal solution to the IP.

- If the optimal solution to the LP relaxation happens to satisfy the integer domain restrictions in the IP, then that is the optimal IP solution.

- Otherwise, the optimal objective value for the LP relaxation gives a bound for the optimal objective value of the IP:
  - For a max IP, the optimal objective value of the LP relaxation gives an upper bound for the optimal objective value of the IP.
  - For a min IP, it is a lower bound.

Idea of branch-and-bound for IP:

1. Solve the LP relaxation.

2. If the optimal solution to the LP relaxation satisfies the integer domain restrictions, then that is the optimal solution to the IP.

3. Otherwise, split the problem into two subproblems by adding a pair of constraints that each exclude the optimal LP solution but do not both exclude any feasible (integer) solution to the IP.

4. Solve the LP relaxations of these subproblems to get <u>bounds</u> for the optimal IP solutions.

5. Repeat this process to explore the subproblem with the best bound.

6. We can stop when:
   - We have a feasible integer solution

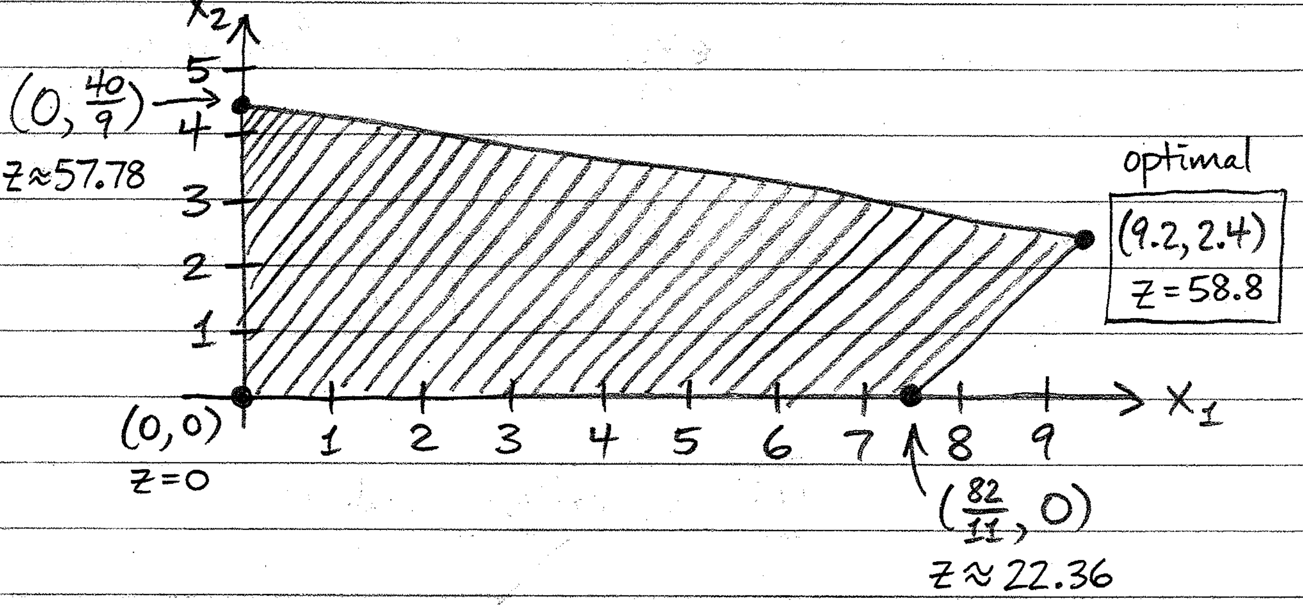
     <u>AND</u>
   - All unexplored subproblems have bounds that show they cannot have integer solutions better than the one we have.

25 June.

Example. $\max\ 3x_1 + 13x_2$

$$\begin{aligned} \text{s.t. } 2x_1 + 9x_2 &\le 40 \\ 11x_1 - 8x_2 &\le 82 \\ x_1 \ge 0,\ x_2 &\ge 0 \\ x_1, x_2 &\text{ integer} \end{aligned}$$

Feasible region of LP relaxation:



But $(9.2, 2.4)$ is not feasible in the IP, because the values of x_1 and x_2 are not integers.

So we will choose an integer variable that has a noninteger value in the optimal LP solution: say, x_2.

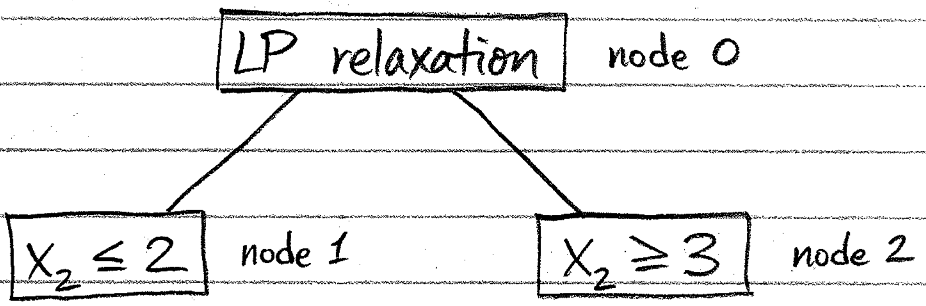
We will <u>branch</u> on the variable x_2.

In the optimal LP solution, $x_2 = 2.4$. We exclude this noninteger value by reasoning as follows:

In any feasible solution to the IP, x_2 must be an integer, so either $x_2 \leq 2$ or $x_2 \geq 3$.

This reasoning excludes all noninteger values like 2.4 that lie between 2 and 3, but it does not exclude any feasible solutions to the IP.

We can represent this reasoning graphically with a <u>branch-and-bound tree:</u>

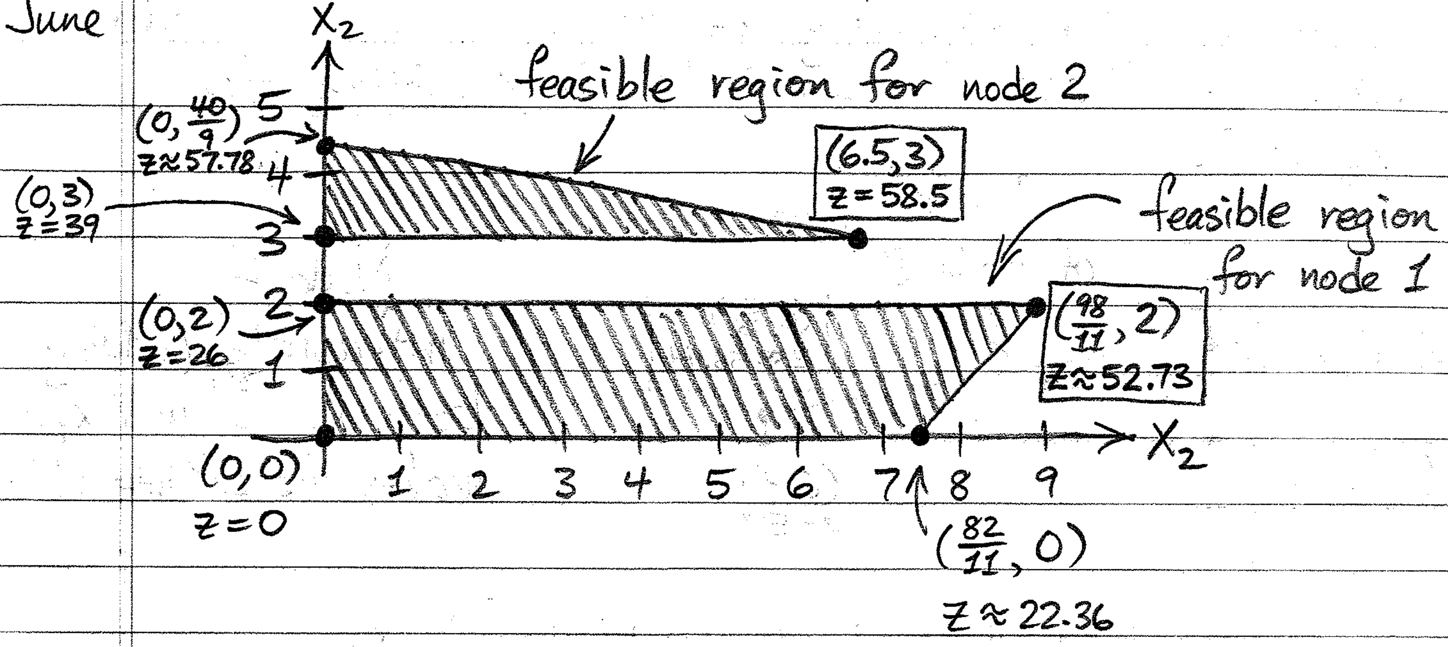


This node represents the LP relaxation with the added constraint $x_2 \leq 2$.

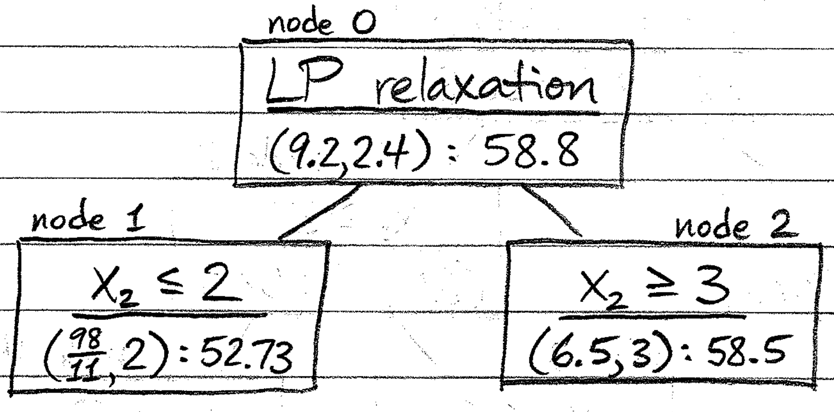
This node represents the LP relaxation with the added constraint $x_2 \geq 3$.

25 June

Branch-and-bound example — ②



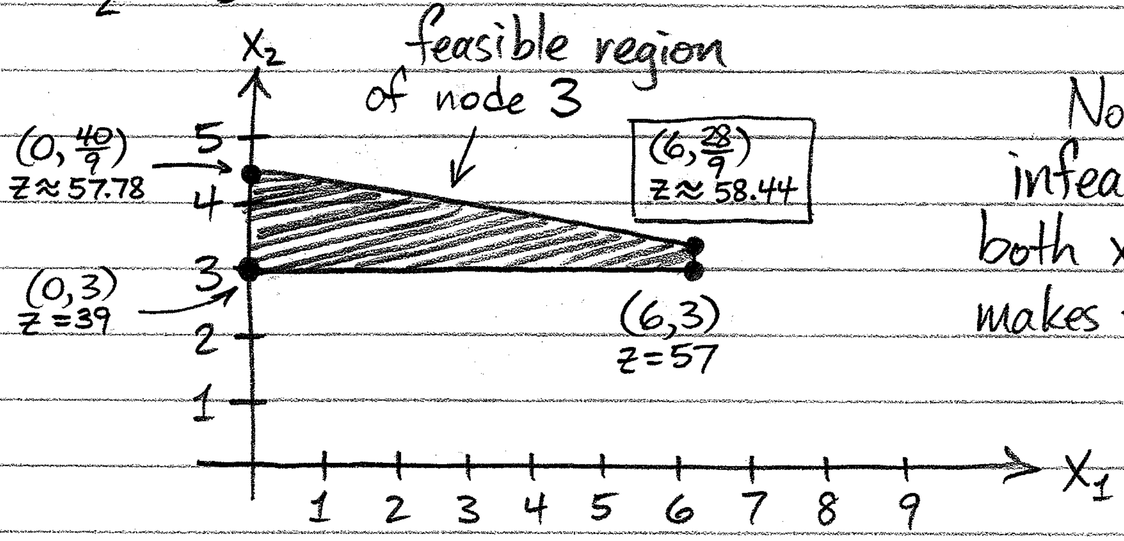
It is useful to record optimal solutions and objective values in the branch-and-bound tree.



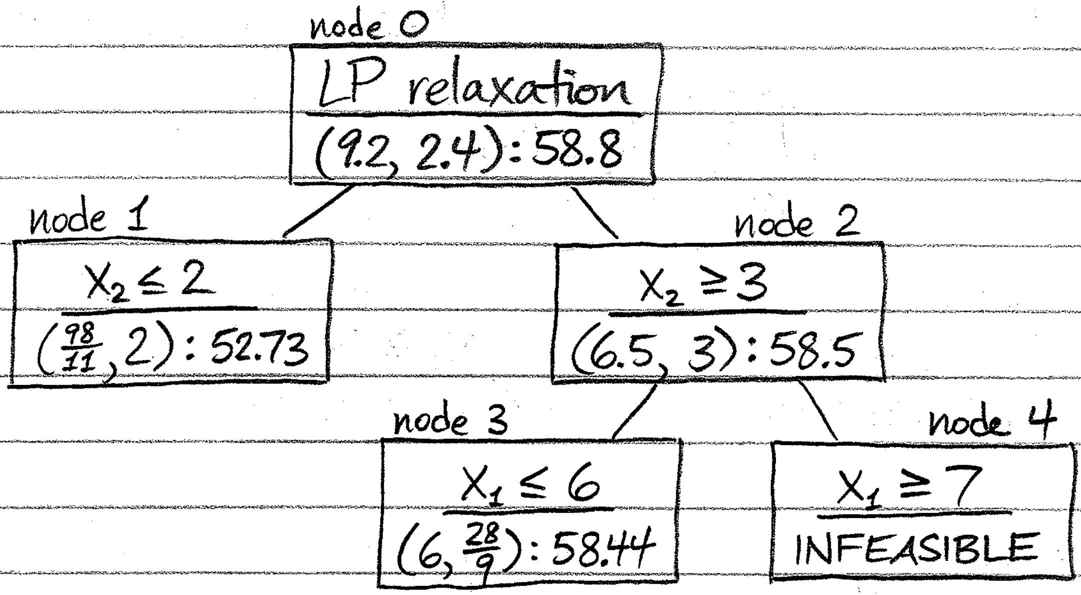
We do not yet have an integer solution — both of the optimal solutions to these subproblems have noninteger values for x_1. So we must branch again.

Node 2 seems most promising — we were able to achieve an objective value of 58.5 there. So we will explore below node 2 next, branching on x_1 to exclude the value 6.5.

Our branches are $x_1 \leq 6$ or $x_1 \geq 7$. Since we are exploring below node 2, we add each of these constraints in addition to $x_2 \geq 3$.



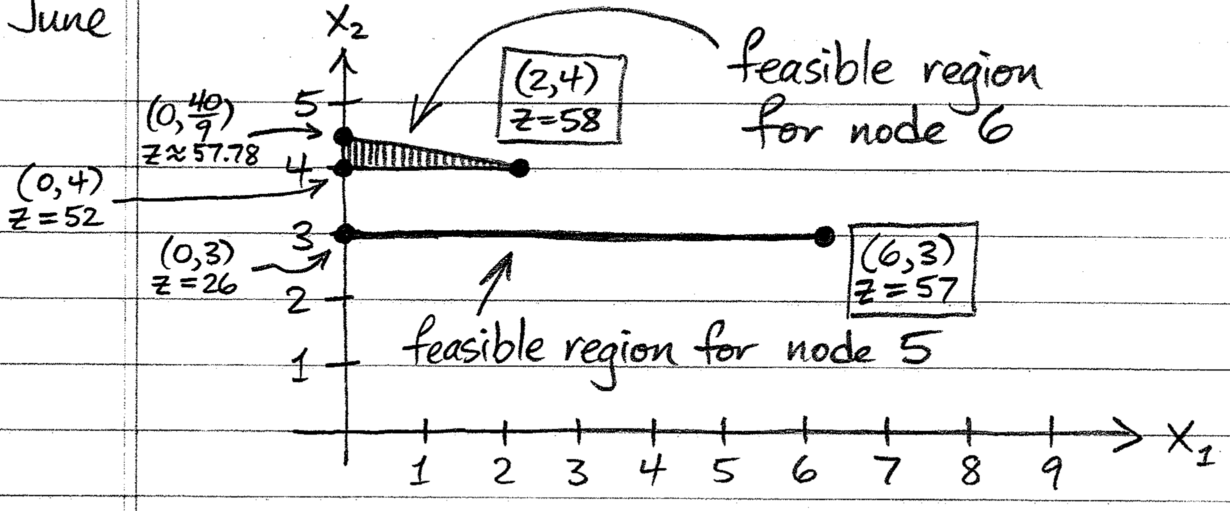
Node 4 is infeasible (adding both $x_2 \geq 3$ and $x_1 \geq 7$ makes the LP infeasible).

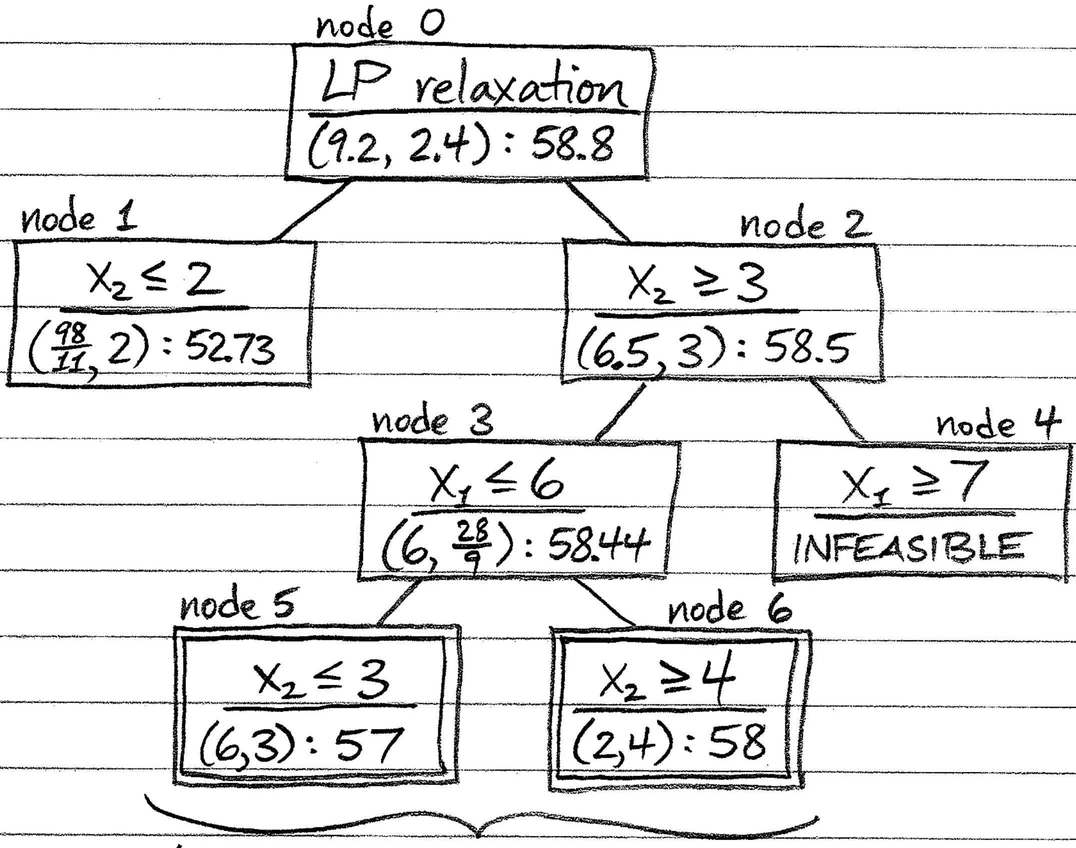


We still do not have an integer solution. We will explore from node 3 next, as it appears to be the most promising. Since x_2 has a noninteger value in the optimal LP solution in node 3, we will branch on x_2. Our branches are $x_2 \leq 3$ or $x_2 \geq 4$.

25 June

Branch-and-bound example — ③





At these nodes, optimal LP solution is feasible in IP (all variables are integers).

We have not explored below node 1, but the bound 52.73 shows that we cannot possibly get an integer solution with objective value greater than 52.73. So we can skip exploring the subtree below node 1 (it has been pruned) and conclude that (2,4) is OPTIMAL!

25 June

A branch-and-bound algorithm for the knapsack problem.

Recall: Knapsack problem.

(Weights can also be viewed as costs.)

Input: A finite set of n items, each having a corresponding weight w_i and value v_i; and a capacity W.

Output: A subset of the items that maximizes total value without the total weight exceeding the capacity.

IP formulation:

Variables $x_i \in \{0,1\}$: whether to include the corresponding item.

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\} \text{ for all } i.$$

LP relaxation:

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\text{s.t} \sum_{i=1}^{n} w_i x_i \leq W$$

$$0 \leq x_i \leq 1 \text{ for all } i.$$

The LP relaxation is a formulation for the fractional knapsack problem, in which we can take fractions of items, with the corresponding fractional weights and values.

The LP relaxation is easy to solve: Sort the items by desirability (value/weight), and then greedily take items in decreasing order of desirability, possibly including a fraction of some item to reach the capacity exactly.

- So we can easily compute bounds for nodes in a branch-and-bound algorithm.

8.2 The knapsack problem

In this section we consider a mathematical programming problem with a structure leading to a natural introduction to the branch-and-bound approach referred to in the introduction. The general form of the *knapsack problem* involving a choice of n items is given below:

$$\begin{array}{lll} \text{(KP)} & \text{Maximize :} & v_1x_1 + v_2x_2 + \cdots + v_nx_n \\ & \text{Subject to :} & w_1x_1 + w_2x_2 + \cdots + w_nx_n \le W \\ & & x_j \in \{0,1\},\ j = 1, 2, \ldots, n. \end{array}$$

The motivation for the name of the problem is planning for a hiking trip in which there is a limit to the weight that can be carried as well as a desire to take the most useful set of items. The v_j's are the *values* of the corresponding items, and the w_j's are their *weights* or *costs*. In the suggested setting of packing a knapsack, the value of an item is its estimated utility, and the cost is its weight, hence the use of the w_j's in the problem. The W in the right-hand side of the constraint represents the total weight that can be carried, or the total budget for a problem involving finances. The restriction of the value of x_j to zero or one indicates the use of x_j to reflect a decision: $x_j = 1$ means that item j is put into the knapsack; $x_j = 0$ means that it is not included.

The number of solutions to a knapsack problem increases rapidly with the number of items. Since a solution is determined by the subset of the n items that is included in the knapsack, the number of solutions – not necessarily all feasible – is 2^n, the number of subsets of an n-element set.

Hence, a problem with five items has $2^5 = 32$ possible solutions, while a problem with ten items has $2^{10} = 1,024$ possible solutions. An effective algorithm for the knapsack problem should obtain the optimal solution by examining only a small fraction of these solutions.

To introduce what we mean by a branch-and-bound algorithm, consider the graph in Figure 8.2.1.

In the tree associated with the solution of a problem by a branch-and-bound process, each node will correspond to a feasible solution, or partial solution, to the problem. A node in a tree is called a *terminal node* if only one edge is incident on the node. The edges indicate a relationship in which a choice between two alternatives has been made leading to two nodes closer to a complete solution. The process will begin at the node labeled R, for the *root* of the tree. We then devise a way to choose between alternatives, and a way to calculate a bound for the maximum (or minimum) value of all solutions further out on the tree. The process will end when a solution has been reached at a terminal node and the bounds indicate that all other solutions are less desirable.

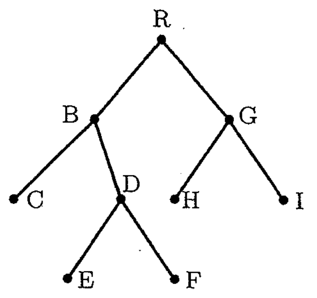


Figure 8.2.1

We will develop the solution process in the context of an example involving real estate.

Example 8.2.1. A real estate development firm is considering five projects for which it can raise an estimated $100 million in capital. For each project, the firm's analysts have produced an estimate of the return of the project over the next 20 years. The estimated returns and project costs in millions of dollars are shown in Table 8.2.1.

Table 8.2.1

Project	Cost	Return
1	45	90
2	28	60
3	32	56
4	18	42
5	24	60

Before describing the algorithm to be used for the knapsack problem, we first consider what is necessary to define such an algorithm. Four things are needed in the description of a branch-and-bound algorithm:

- The order in which to consider the branches,
- The rule that terminates the search for the solution,
- The decision which forms the basis for a branch, and

- The formula for the calculation of the bound at each node of the search tree.

In our algorithm for the knapsack problem:

- The items will be considered in decreasing order by their *desirability*, i.e., the ratio of their value to their cost,
- The algorithm will terminate when all items have either been selected or eliminated to form a complete solution, and any other selection has been shown to have a value that is not greater,
- A branch will be determined by the decision to include or not include an item, and
- The bound will be the sum of the values of the items selected plus a high estimate of the value of those items that might still be selected.

Branch-and-bound algorithm for the knapsack problem

We first restate the problem and establish the notation for the algorithm:

$$\begin{array}{lrl} \text{(KP)} & \text{Maximize :} & v_1x_1 + v_2x_2 + \cdots + v_nx_n \\ & \text{Subject to :} & w_1x_1 + w_2x_2 + \cdots + w_nx_n \leq W \\ & & x_j \in \{0,1\},\ j = 1,2,\ldots,n. \end{array}$$

In the statement of the algorithm, we will let S denote the set of all n items, B_i be the bound at node i, I_i be the set of items included at node i, E_i be the set of items excluded at node i, and n be the current number of nodes. The empty set is denoted by $\emptyset$.

We want the bound at a node corresponding to an infeasible solution to be an arbitrarily large negative number to be certain that any other bound is greater. For this purpose we use a convention called the "Big M." Here M denotes a positive number with the property that $M - a$ is positive no matter how large a may be. In this instance, since we want to express a large negative number, we will set the bound equal to $-M$. This same convention will be used in later branch-and-bound algorithms.

1. **Check feasibility:**

   (a) Check that j exists such that $w_j \leq W$. If not, there is **no feasible solution** and the algorithm terminates.

(b) If $\sum_{j=1}^{n} w_j \leq W$, then all items can be loaded, and the problem is **trivially solved** by putting $I_1 = S$ and $E_1 = \emptyset$.

(c) If the algorithm did not terminate at (a) or (b), **sort** the items into decreasing order of desirability $\frac{v_i}{w_i}$.

(d) Create **initial node:** Set $n = 1, B_n = \sum_{j=1}^{n} v_j$, and $I_n = E_n = \emptyset$.

2. **Select the node for the next branch:**

   (a) Select the terminal node k with B_k the largest existing bound for the next branching decision.

   (b) If $I_k \cup E_k = S$, then an **optimal solution** is given by the items in I_k, so **stop**. Otherwise, form the next branch on the remaining item with index i' having the **largest desirability**

$$\frac{v_{i'}}{w_{i'}} = \max\left\{\frac{v_j}{w_j} : j \notin I_k \cup E_k\right\}.$$

3. **Form the next two nodes:**

   (a) Set $n = n+1$, and index the **new left node** by n, and set $I_n = I_k$ and $E_n = E_k \cup \{i'\}$ to **exclude** item i'.

   (b) Set $n = n + 1$, and index the **new right node** by n, and set $I_n = I_k \cup \{i'\}$ and $E_n = E_k$ to **include** item i'.

4. **Calculate the upper bounds of the new nodes:**

   (a) If $\sum\{w_j : j \in I_k\} > W$, then the node corresponds to an **infeasible solution**, so set $B_k = -M$.

   If $W - \sum\{w_j : j \in I_k\} < w_j$ for all $j \notin I_k \cup E_k$, then set $B_k = \sum\{v_j : j \in I_k\}$.

   Otherwise, let B_k be $\sum\{v_j : j \in I_k\}$ plus the sum of the values v_j of all items in $S\backslash(I_k \cup E_k)$, taking them in order of decreasing desirability until the next item to be added would make the total of the weights greater than W. Add to B_k the proportional part of the next item considered if a fractional part of it can be included.

   (b) When (a) has been applied for both new nodes, go to Step 2.

We now solve Example 8.2.1. Since there is at least one project that can be done within the budget, the problem has a feasible solution. Since the total of all the costs is more than the budget, there is no trivial solution.

We therefore sort the projects in decreasing order by desirability:

Table 8.2.2

Project	Cost	Return	Desirability
5	24	60	2.50
4	18	42	2.33
2	28	60	2.14
1	45	90	2.00
3	32	56	1.75

Because project 5 has the greatest desirability, we select it for the first branch. In the new left node, Node 2, we exclude project 5. We can include in the bound the costs of projects 4, 2, and 1 since the total of their costs is 91. When a fractional part of the return on project 3 is included to bring the total cost to 100, the bound is

$$B_2 = 42 + 60 + 90 + \frac{9}{32} \cdot 56 = 207.75.$$

In the new right node, Node 3, we include project 5. Since the total of the costs of projects 5, 4, and 2 is 70, we can include the total of their returns in the bound plus a proportional part of the return for project 1:

$$B_3 = \underline{60} + 42 + 60 + \frac{30}{45} \cdot 90 = 222.$$

In the calculation of B_3 we have underlined 60 to indicate that it is the value of an included item. This notational convention can help to identify when it is necessary to include a fractional part of the cost of an item.

Note in the initial tree, shown in Figure 8.2.2, that we use an * to indicate an excluded project.

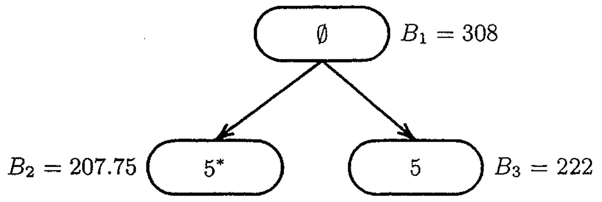


Figure 8.2.2

Since Node 3 is the terminal node with the highest bound, we branch from it to form the new nodes. We branch on the decision to include project 4 since it has the highest desirability of the nodes remaining to consider.

For the left node, Node 4, we exclude project 4. The sum of the costs of projects 5, 2, and 1 is 97, so the bound is:

$$B_4 = \underline{60} + 60 + 90 + \frac{3}{32} \cdot 56 = 215.25.$$

The new right node includes project 4, and since the costs of projects 5, 4, and 2 sum to 70, the bound for Node 5 is:

$$B_5 = \underline{60} + \underline{42} + 60 + \frac{30}{45} \cdot 90 = 222.$$

Now branching from Node 5 with the decision based on project 2, we calculate the bounds as follows:

$$\begin{aligned} B_6 &= \underline{60} + \underline{42} + 90 + \frac{13}{32} \cdot 56 = 214.75 \\ B_7 &= \underline{60} + \underline{42} + \underline{60} = 162. \end{aligned}$$

Note that the calculation of B_7 omits the fractional part of the next return because the total cost of the projects included is 70 and no other project could be included since the budget is only 100 and the smallest remaining project has a cost of 32.

The resulting tree is shown in Figure 8.2.3.

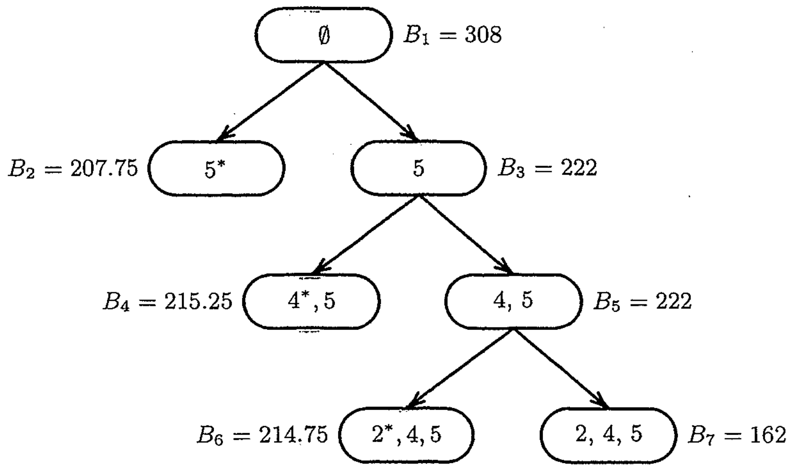


Figure 8.2.3

The largest bound on a terminal node is $B_4 = 215.25$. Since not all projects are included or excluded at Node 4, we have not reached a solution, but must continue branching from that node. We branch on the decision to include or exclude project 2, and continue with the calculations summarized below to achieve the final search tree:

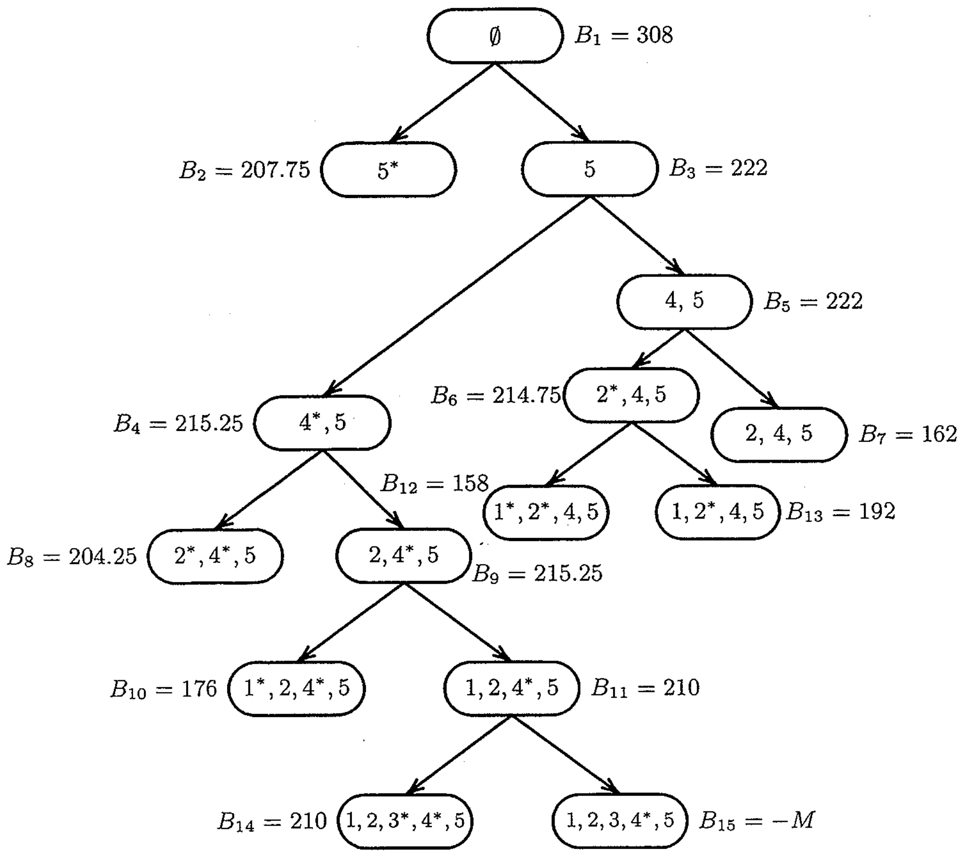
$$\begin{array}{lll}
I_8 = \{5\}, & E_8 = \{2,\ 4\}, & B_8 = \underline{60} + 90 + \frac{31}{32} \cdot 56 = 204.25 \\
I_9 = \{2,\ 5\}, & E_9 = \{4\}, & B_9 = \underline{60} + \underline{60} + 90 + \frac{3}{32} \cdot 56 = 215.25 \\
I_{10} = \{2, 5\}, & E_{10} = \{1, 4\}, & B_{10} = \underline{60} + \underline{60} + 56 = 176 \\
I_{11} = \{1, 2, 5\}, & E_{11} = \{4\}, & B_{11} = \underline{60} + \underline{60} + \underline{90} = 210 \\
I_{12} = \{4, 5\}, & E_{12} = \{1, 2\}, & B_{12} = \underline{60} + \underline{42} + 56 = 158 \\
I_{13} = \{1, 4, 5\}, & E_{13} = \{2\}, & B_{13} = \underline{90} + \underline{42} + \underline{60} = 192 \\
I_{14} = \{1, 2, 5\}, & E_{14} = \{3, 4\}, & B_{14} = \underline{60} + \underline{60} + \underline{90} = 210 \\
I_{15} = \{1, 2, 3, 5\}, & E_{15} = \{4\}, & B_{15} = -M
\end{array}$$

Note that for Node 15, projects 1, 2, 3, and 5 are included with a total cost of 129, more than the maximum of 100. Thus, it represents an infeasible solution, so $B_{15} = -M$.

The final graph for the problem is given in Figure 8.2.4 and shows that the optimal solution is found at Node 14. Thus, the firm's best strategy is to carry out projects 1, 2, and 5 at a cost of \$97 million with an expected return of \$210 million.

Note from the numbering of the nodes that a bound not the highest on a terminal node at one stage may become the highest on a terminal node at a later stage of the search. ■

The inclusion of the fractional part of an item in the calculation of the bounds is necessary. An example in the exercises shows that omitting the fractional part can cause the algorithm to fail to detect the branch of the tree containing the solution because the bound is too low.



∅ $B_1 = 308$
$B_2 = 207.75$ 5^*
5 $B_3 = 222$
4, 5 $B_5 = 222$
$B_4 = 215.25$ $4^*, 5$
$B_6 = 214.75$ $2^*, 4, 5$
2, 4, 5 $B_7 = 162$
$B_{12} = 158$ $1^*, 2^*, 4, 5$
$1, 2^*, 4, 5$ $B_{13} = 192$
$B_8 = 204.25$ $2^*, 4^*, 5$
$2, 4^*, 5$ $B_9 = 215.25$
$B_{10} = 176$ $1^*, 2, 4^*, 5$
$1, 2, 4^*, 5$ $B_{11} = 210$
$B_{14} = 210$ $1, 2, 3^*, 4^*, 5$
$1, 2, 3, 4^*, 5$ $B_{15} = -M$

Figure 8.2.4

Exercises

1. Solve the knapsack problem if the maximum possible total weight is 40:

Table 8.2.3

	Weight	Value
1	15	20
2	12	24
3	9	15
4	8	14

2. If a budget is \$20 and the possible purchases are tabled below, determine which to purchase: